**Prerequisites: Setting Up Your Environment**

First, let's create the tables and insert some sample data for the three scenarios. You'll need tables for Accounts, Employees, and Departments.

**Run these CREATE TABLE and INSERT statements first:**

SQL

-- =========== SETUP FOR SCENARIO 1 & 3 ===========

CREATE TABLE Accounts (

AccountID NUMBER(10) PRIMARY KEY,

CustomerID NUMBER(10),

AccountType VARCHAR2(20) CHECK (AccountType IN ('Savings', 'Checking')),

Balance NUMBER(12, 2) CHECK (Balance >= 0) -- Constraint to prevent negative balance

);

-- Insert sample accounts

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance) VALUES (1001, 1, 'Savings', 5000.00);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance) VALUES (1002, 1, 'Checking', 1200.00);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance) VALUES (2001, 2, 'Savings', 8500.50);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance) VALUES (3001, 3, 'Savings', 120000.00);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance) VALUES (3002, 3, 'Checking', 25000.00);

-- =========== SETUP FOR SCENARIO 2 ===========

CREATE TABLE Departments (

DepartmentID NUMBER(4) PRIMARY KEY,

DepartmentName VARCHAR2(30)

);

CREATE TABLE Employees (

EmployeeID NUMBER(6) PRIMARY KEY,

FirstName VARCHAR2(20),

LastName VARCHAR2(25),

DepartmentID NUMBER(4) REFERENCES Departments(DepartmentID),

Salary NUMBER(8,2)

);

-- Insert sample departments and employees

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES (10, 'Administration');

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES (20, 'Marketing');

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES (50, 'Shipping');

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES (80, 'Sales');

INSERT INTO Employees (EmployeeID, FirstName, LastName, DepartmentID, Salary) VALUES (101, 'John', 'Smith', 20, 60000);

INSERT INTO Employees (EmployeeID, FirstName, LastName, DepartmentID, Salary) VALUES (102, 'Jane', 'Doe', 20, 65000);

INSERT INTO Employees (EmployeeID, FirstName, LastName, DepartmentID, Salary) VALUES (103, 'Peter', 'Jones', 80, 100000);

INSERT INTO Employees (EmployeeID, FirstName, LastName, DepartmentID, Salary) VALUES (104, 'Mary', 'Williams', 80, 110000);

INSERT INTO Employees (EmployeeID, FirstName, LastName, DepartmentID, Salary) VALUES (105, 'David', 'Brown', 50, 45000);

COMMIT;

**Exercise 3: Stored Procedure Solutions**

**Scenario 1: Process Monthly Interest**

**Question:** Write a stored procedure ProcessMonthlyInterest that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

**Solution:**

This procedure uses a single, efficient UPDATE statement to modify all rows that match the criteria (AccountType = 'Savings').

SQL

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

-- No parameters are needed as the rate is fixed at 1% for all savings accounts

BEGIN

-- Update all savings accounts by increasing the balance by 1%

UPDATE Accounts

SET Balance = Balance \* 1.01 -- (Balance + Balance \* 0.01)

WHERE AccountType = 'Savings';

-- The COMMIT statement saves the changes permanently.

-- It's good practice to commit inside the procedure if it represents a complete transaction.

COMMIT;

-- Optional: Log a message to the console

DBMS\_OUTPUT.PUT\_LINE('Monthly interest of 1% processed for ' || SQL%ROWCOUNT || ' savings accounts.');

EXCEPTION

WHEN OTHERS THEN

-- If any error occurs, rollback all changes

ROLLBACK;

-- Log the error

DBMS\_OUTPUT.PUT\_LINE('An error occurred during interest processing: ' || SQLERRM);

-- Re-raise the exception to notify the calling application

RAISE;

END ProcessMonthlyInterest;

/

**How to Execute the Procedure:**

SQL

-- Make sure to enable server output to see the message

SET SERVEROUTPUT ON;

BEGIN

ProcessMonthlyInterest;

END;

/

-- You can verify the changes by selecting the data

SELECT \* FROM Accounts WHERE AccountType = 'Savings';

**Scenario 2: Update Employee Bonus**

**Question:** Write a stored procedure UpdateEmployeeBonus that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

**Solution:**

This procedure accepts two parameters: the department ID and the bonus rate. It uses these parameters in the UPDATE statement's WHERE and SET clauses, respectively.

SQL

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department\_id IN Employees.DepartmentID%TYPE,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

-- Check for invalid bonus percentage

IF p\_bonus\_percent <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Bonus percentage must be a positive number.');

END IF;

-- Update salaries for all employees in the specified department

-- The bonus is added to the existing salary.

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_percent / 100)

WHERE DepartmentID = p\_department\_id;

-- If no rows were updated, it might mean the department ID was invalid.

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Warning: No employees found for Department ID ' || p\_department\_id || '. No salaries were updated.');

ELSE

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT || ' employees in Department ' || p\_department\_id || ' received a ' || p\_bonus\_percent || '% bonus.');

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error applying bonus: ' || SQLERRM);

RAISE;

END UpdateEmployeeBonus;

/

**How to Execute the Procedure:**

SQL

SET SERVEROUTPUT ON;

-- Give a 5% bonus to the 'Sales' department (ID 80)

BEGIN

UpdateEmployeeBonus(p\_department\_id => 80, p\_bonus\_percent => 5);

END;

/

-- Give a 10% bonus to the 'Marketing' department (ID 20)

BEGIN

UpdateEmployeeBonus(20, 10); -- You can also pass parameters by position

END;

/

-- Verify the changes

SELECT EmployeeID, FirstName, DepartmentID, Salary FROM Employees ORDER BY DepartmentID;

**Scenario 3: Transfer Funds Between Accounts**

**Question:** Write a stored procedure TransferFunds that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

**Solution:**

This is a classic transaction processing example. The procedure ensures that the entire operation (debit from source, credit to destination) succeeds or fails as a single unit. It locks the source account row to prevent other sessions from modifying it during the transaction.

SQL

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_source\_account\_id IN Accounts.AccountID%TYPE,

p\_dest\_account\_id IN Accounts.AccountID%TYPE,

p\_transfer\_amount IN Accounts.Balance%TYPE

) IS

v\_source\_balance Accounts.Balance%TYPE;

BEGIN

-- Step 1: Validate the transfer amount

IF p\_transfer\_amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Transfer amount must be positive.');

END IF;

-- For safety, prevent transferring to the same account

IF p\_source\_account\_id = p\_dest\_account\_id THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Source and destination accounts cannot be the same.');

END IF;

-- Step 2: Lock the source account row and check its balance.

-- SELECT ... FOR UPDATE locks the row until a COMMIT or ROLLBACK is issued.

-- This prevents race conditions where another transaction could alter the balance

-- after we check it but before we update it.

SELECT Balance INTO v\_source\_balance

FROM Accounts

WHERE AccountID = p\_source\_account\_id

FOR UPDATE;

-- Step 3: Check for sufficient funds

IF v\_source\_balance < p\_transfer\_amount THEN

-- If not enough funds, raise an error. The ROLLBACK will release the lock.

RAISE\_APPLICATION\_ERROR(-20004, 'Insufficient funds in source account ' || p\_source\_account\_id);

ELSE

-- Step 4: If funds are sufficient, proceed with the transfer

-- Debit the source account

UPDATE Accounts

SET Balance = Balance - p\_transfer\_amount

WHERE AccountID = p\_source\_account\_id;

-- Credit the destination account

UPDATE Accounts

SET Balance = Balance + p\_transfer\_amount

WHERE AccountID = p\_dest\_account\_id;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful.');

-- Step 5: Commit the transaction

COMMIT;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK; -- Release any locks

DBMS\_OUTPUT.PUT\_LINE('Error: One of the account IDs is invalid.');

RAISE;

WHEN OTHERS THEN

ROLLBACK; -- Rollback on any other error

DBMS\_OUTPUT.PUT\_LINE('An error occurred during the transfer: ' || SQLERRM);

RAISE;

END TransferFunds;

/

**How to Execute the Procedure:**

SQL

SET SERVEROUTPUT ON;

-- Check balances before transfer

SELECT AccountID, Balance FROM Accounts WHERE AccountID IN (1001, 1002);

-- Scenario A: Successful Transfer

-- Transfer $500 from account 1001 to 1002

BEGIN

TransferFunds(

p\_source\_account\_id => 1001,

p\_dest\_account\_id => 1002,

p\_transfer\_amount => 500

);

END;

/

-- Check balances after successful transfer

SELECT AccountID, Balance FROM Accounts WHERE AccountID IN (1001, 1002);

-- Scenario B: Failed Transfer (Insufficient Funds)

-- Try to transfer $10000 from account 1001 (which doesn't have enough)

BEGIN

TransferFunds(1001, 1002, 10000);

END;

/

-- Note: This will raise an ORA-20004 error, which is expected.

-- Check balances to confirm no changes were made

SELECT AccountID, Balance FROM Accounts WHERE AccountID IN (1001, 1002);